PVM Support for Clusters

Paul L. Springer

Jet Propulsion Laboratory

California Institute of Technology

4800 Oak Grove Drive, 168-522

Pasadena, CA 91109

Paul.Springer@jpl.nasa.gov

Abstract. The latest version of PVM (3.4.3) now contains support for a PC cluster running Linux, also known as a Beowulf system. A PVM user of a computer outside the cluster can add the cluster as a single machine. All details of the cluster are hidden, and spawning to the cluster automatically puts tasks on individual nodes. This will work even if the nodes are on a private network, as long as the user's computer has a way of connecting to the "front-end" of the cluster.

1 Introduction

One of the big advantages PVM offers is its ability to easily tie together heterogeneous systems. When one of those systems is a cluster, complications can arise. If PVM sees the cluster as a collection of individual computers, each with its own connectivity to the rest of the PVM network, the cluster can't be treated as a single system image, and instead the user must treat the cluster more as a distributed

system. This has the disadvantage that the user's application must issue PVM *addhost* and *spawn* commands individually for each node in the cluster. Furthermore, each node incurs the overhead of running the PVM daemon.

If the cluster's nodes can't be accessed from outside the cluster, then PVM running on an external computer can not make full use of the cluster. This is due to the fact that PVM's *addhost* command requires a name that is addressable by the external computer. The latest version of PVM, 3.4.3, includes new features to address these limitations.

2 Design

A number of ports to different computers are included with the PVM release[1]. Some of these are for massively parallel processors (MPP's), such as the PGON port for the Intel Paragon and the SP2MPI port for IBM's SP-2. Cluster support in PVM has been implemented as an MPP port, and given the architecture name of BEOLIN.

The BEOLIN port is modeled after the SP2MPI port, substituting the use of sockets for MPI as the underlying communication layer. When PVM is initiated on the cluster, the PVM daemon (pvmd) starts up on the front-end node. (The front-end node is the one which can communicate to computers outside of the cluster.) An environment variable is read by pvmd that tells it what cluster nodes are available for use. Each subsequent spawning request that arrives causes a node to be allocated to a single task (see Figure 1). The daemon marks the target nodes as busy, so that no two tasks will share the same node. When a task is finished, the node is released back into the node pool for later use. If the user attempts to spawn more tasks than there are

nodes, PVM will return an “Out of Resources” error. A single spawn of multiple copies of a task is allowed. In this case each copy of the task operates on its own node.

When a task is spawned to a node, pvmd forks, and the child process invokes rsh to initiate the task on the assigned node. The standard output and standard error of each remote task are connected to the corresponding child process. Each such child process is visible to pvmd, and the pvm monitor will show them as tasks with the suffix “.host” appended to the task name. The pvmd daemon is not invoked on any but the front-end node.

Once a task starts on a node, it looks in the /tmps file system (shared with the front-end) for a file name beginning with “pvmd” and ending with the user-id. It uses the information in that file to determine the IP address and port of the front-end, and then makes a socket connection to the pvmd daemon. PVM messages and commands are then transferred back and forth through this connection.

3 Installing and Using the BEOLIN Port of PVM

In PVM 3.4.3, BEOLIN is a defined architecture. Before building or running this port, the user must define the environment variable **PVM_ARCH** to have the value **BEOLIN**. Do not rely on the **pvmgetarch** program to return this value—**PVM_ARCH** must be set using the **setenv** command or equivalent. If this is not done, in all likelihood the user will end up with the standard LINUX port of PVM instead. Once this variable has been defined, PVM can be built in the normal way (see Chapter 9 of [1]).

In order for the BEOLIN port of PVM to run, the cluster must have certain capabilities. A /tmps subdirectory shared by all nodes and the front-end must be set up. This can be as simple as a symbolic link to a previously defined shared file system. Furthermore, the nodes must be able to connect to the front-end using the address returned by gethostname() on the front-end.

The pvmd on the front-end determines which nodes are available and how to address them by examining the environment variable **PROC_LIST**. This should be defined in the user's front-end environment, typically in the .cshrc file. The value of the variable should be set to a colon separated list of the names of the nodes. For example, if the nodes available are named n0, n1, n2, and n3, the corresponding line in the .cshrc file would be **setenv PROC_LIST n0:n1:n2:n3**.

To avoid the possibility of the pvmd becoming a bottleneck in passing messages between nodes, it is strongly recommended that direct message routing be used, by calling the **pvm_setopt()** routine with the **PvmRouteDirect** parameter. This will allow messages from one node to another to go directly, instead of being routed through the daemon.

Restrictions

There are certain restrictions in the BEOLIN port of PVM. The PVM monitor may be run from the front-end, and programs spawned to the nodes from there. However, running the monitor or pvmd on a node other than the front-end has not been extensively tested.

The pvm_addhosts() call should be used by the application only to add the front-end node of the cluster. It should not be used to add other nodes to the virtual machine. With this port, the entire Beowulf system is made to look like a single individual computer, and the individual nodes should not be addressed. For the same reason, do not use the PVM monitor **add** command to add individual nodes, and do not include node names other than the front-end's name in any hostfile used as an argument when starting PVM.

If all nodes of the cluster have direct connectivity to the external network, this port can still be used with the cluster. In this case, if there is no obvious node to designate as the front-end node, any single node can be arbitrarily designated as such.

Under the version of LINUX used to develop this port, a terminate signal would not propagate to a remote process initiated by rsh. This prevented the **pvm_kill()** command from working correctly. It may be that later versions of LINUX will solve this problem.

4 Applications

The impetus for developing this port of PVM came from an effort to enable the Matpar program [2] to work on a Beowulf cluster. The Matpar software provides parallel extensions to MATLAB, a popular commercial program used by scientists and engineers to perform matrix operations. It enables the user to apply parallelism to certain MATLAB functions in order to speed up the operation of those functions.

An important goal in the design of Matpar was to make the software very easy to use for engineers who may be unfamiliar with parallel computers and who do not

have the time to learn a new way of doing things. With Matpar the user does not need to learn how to compile a program, nor does he need to learn how to move his data and program to a parallel computer. The user is also not required to coordinate other copies of MATLAB running on other nodes, and does not need to worry about splitting up the task between the parallel processors.

The Matpar software follows a client/server approach. The client software resides on a workstation, and the server software is on a parallel computer (see Figure 2). To meet the requirement of transparency for the workstation user, PVM was chosen because of its sophisticated ability to initiate tasks on remote computers. The first time a parallel equivalent to one of MATLAB's functions is invoked, the client side of the Matpar software is initiated, which in turn uses PVM to start a virtual machine. The client then adds the parallel computer to that virtual machine and spawns the necessary tasks onto the parallel computer. Data and commands are sent back and forth between client and server using PVM message passing services.

Each time the server software on the parallel computer receives a new command with its corresponding data, it calls the appropriate support routines to handle the command. The support software consists primarily of ScaLAPACK library routines. Underlying these are the BLACS communication routines which are in turn built on top of PVM.

At JPL, one of the areas in which Matpar can be of use is in calculating frequency response functions for space interferometer designs. One representative model having 1 output, 6 inputs, and 1160 states was evaluated for 2203 frequency points. The calculations took 26 minutes on a Sun UltraSPARC-II running MATLAB, but only 2 minutes when Matpar was used together with a cluster.

Matpar has been implemented on a Beowulf system in our lab at JPL. This system is not reserved exclusively for batch jobs, but is left open for use on an interactive basis. This, together with the faster calculation times the cluster can offer, brings the capability of interactive supercomputing to the user.

5 Future Work

The largest number of nodes used so far with this port of PVM has been 31. The software needs to be run on much larger machines in order to characterize its performance at the high end. It is unclear what bottlenecks and limitations exist, and how they will manifest themselves, when one pvmd daemon is controlling a large number of tasks. Changes in the pvmd code to accommodate very large systems may be necessary.

As mentioned previously, the pvm_kill() function does not work with the version of Linux used (2.2.2) in the development of the BEOLIN port. This problem needs further investigation.

Interest has been expressed in creating a similar port for other kinds of clusters. Porting to UNIX-like systems should be fairly straightforward. Work has also been accomplished on a port to a Windows NT cluster.

6 Conclusion

The BEOLIN architecture included in PVM 3.4.3 offers new capabilities to the PVM applications programmer. A Linux PC cluster can now be added to the virtual

machine, and parallel tasks spawned onto the cluster, even when the individual nodes are on a private network.

The software described here, together with the relatively low cost of a Beowulf system and the on-demand availability of such a system, can open the door to interactive supercomputing. When a parallel computer is used to reduce the computation time of a complex problem to just a couple minutes, a user is able to focus better on the design problem, and does not face such a large cost in time in order to evaluate variations of his design. It is hard to predict what changes may arise because of the availability of interactive supercomputing, but it should be no surprise if they are as profound as those that arose when desktop computing began to replace batch processing.

7 Acknowledgements

The work described in this report was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

The software described is available for download. The web page for Matpar is http://hpc.jpl.nasa.gov/PS/MATPAR. The web page for PVM is http://www.epm.ornl.gov/pvm

References:

1. Geist, Al, et al: PVM: Parallel Virtual Machine. The MIT Press, Cambridge, Massachusetts (1996)

2. Springer, Paul L., "Matpar Home Page." http://hpc.jpl.nasa.gov/PS/MATPAR

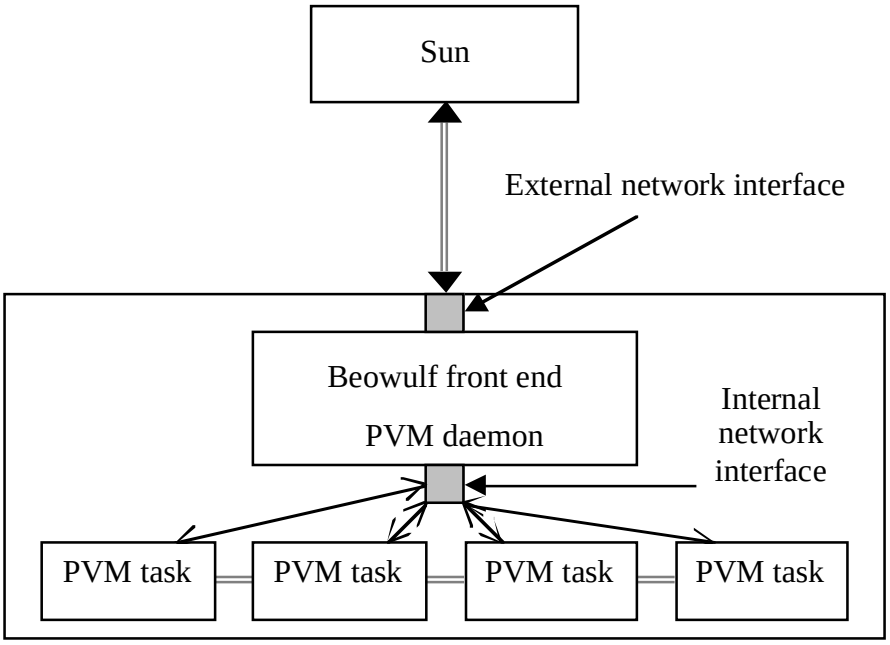

Figure 1. BEOLIN Architecture

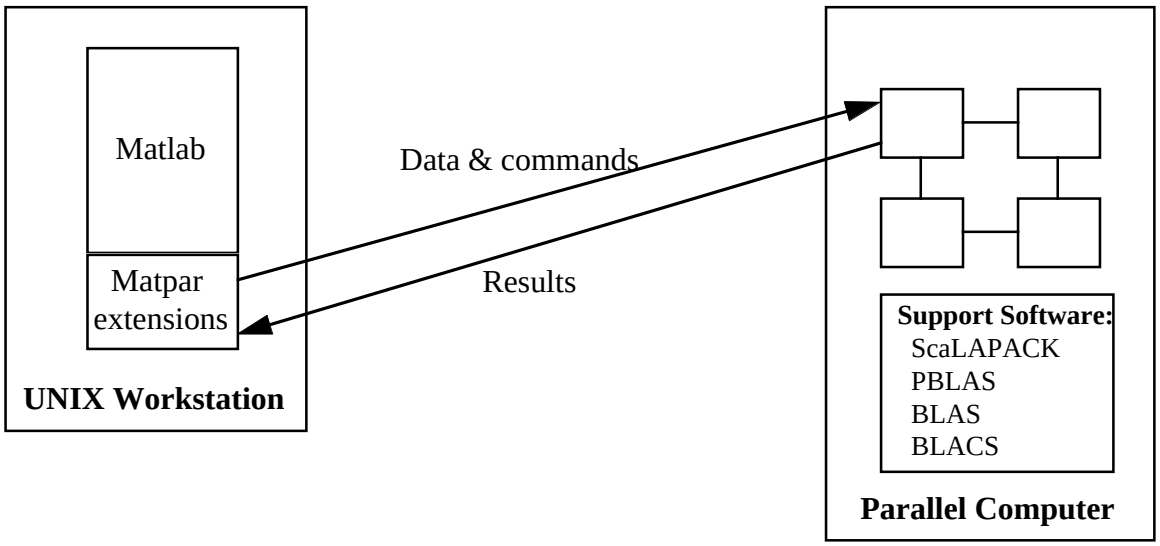

Figure 2. Client/server architecture of Matpar